精通Linux丛书

Linux的内核与编程

雷澍 等编著

关于本书

- 详细讲述了Linux系统实现的原理
- 通过实例介绍了Linux的编程方法及典型的编程工具
- 提供了特定的内核源代码的来源
- 是作者多年从事Unix/Linux编程的经验总结
- 语言通俗易懂，读来轻松自如

机械工业出版社
China Machine Press

本书是精通 Linux 丛书的第三本。本丛书共 3 本，另外两本是《Linux 的安装与使用》和《Linux 的管理与配置》。

本书全面介绍了 Linux 操作系统的内核原理与编程方法，全书分两个部分共 19 章，分别介绍 Linux 操作系统实现的软件基础、内存管理、进程、进程间通信机制、PCI、中断和中断处理、设备驱动器、文件系统、网络、内核机制、模件、内核源代码、内部数据结构、GAWK 语言、C 语言、系统服务、多进程编程、网络程序设计、GTK 编程和字符元编程等内容，可供需要详细了解 Linux 内核原理和开发 Linux 应用程序的中、高级用户使用。

本书内容深入浅出，即使非专业人士也能很容易地理解一些十分专业的概念。

图书在版编目（CIP）数据

Linux 的内核与编程 / 雷澍等编著. －北京：机械工业出版社，2000. 7
（精通 Linux 丛书）

ISBN 7-111-08165-X
Ⅰ. Linux Ⅱ. 雷… Ⅲ. Linux 操作系统-程序设计 Ⅳ. TP316.81
中国版本图书馆 CIP 数据核字（2000）第 64560 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）.
责任编辑：余茂祚 封面设计：艾 迪
责任印制：何全君
中国农业出版社印刷厂印刷·新华书店北京发行所发行
2000 年 7 月第 1 版第 1 次印刷
787mm×1092mm 1/16 · 22.5 印张 · 544 千字
0 001－ 5 000 册
定价：36.00 元

前　言

众所周知，Unix 经过了三十几年的发展完善已经相当可靠、稳定。遗憾的是，Unix 大多运行在昂贵的工作站上，普通人难得一见。 现在，有了 Linux，任何人都可以在家里微机上学习、使用 Unix 了!

Linux 是 Unix 在微机上的完整实现，是芬兰的 Linus Torvalds 于 1991 年独立发展起来 的，由于 Linux 免费提供源代码和可执行文件，并且公布在互联网上，因此从一开始就吸引了世界各地的 Unix 行家为 Linux 编写了大量的驱动程序和应用软件。在短短几年时间里，Linux 就发展成为一个相当完善的操作系统，成为 Unix 世界的一朵奇葩。

Linux 的特点

1．可完全免费得到

Linux 操作系统可以从互联网免费下载使用，只要有快速的网络连接就行； 而且，Linux 上跑的绝大多数应用程序也是可以免费得到的。 用了 Linux 就再也不用背“使用盗版软件”的黑锅了。

2．可以运行在 386 以后的 x86 机器上

Linux 专门为微机环境而设计，充分利用了 X86 CPU 的任务切换能力，使 X86 CPU 的效能发挥得淋漓尽致， 而这一点 Windows 都没有做到。

3．Linux 是 Unix 的完整实现

Unix 上的绝大多数命令都可以在 Linux 里找到并有所加强。Unix 的可靠性、稳定性以及强大的网络功能也在 Linux 身上一一体现。

4．真正的多任务多用户

只有很少的操作系统能提供真正的多任务能力，尽管许多操作系统声明支持多任务，但并不完全准确，如 Windows。而 Linux 则充分利用了 X86 CPU 的任务切换机制，实现了真正多任务、多用户环境，允许多个用户同时执行不同的程序，并且可以给紧急任务以较高的优先级。

5．完全符合 POSIX 标准

这使 Unix 下的许多应用程序可以很容易地移植到 Linux 下，相反也是这样。

6．具有图形用户界面

Linux 的图形用户界面是 X Window 系统。X Window 可以做 MS Windows 下的所有事情， 而且更有趣、更丰富，你甚至可以在几种不同风格的窗口之间来回切换。

7．具有强大的网络功能

实际上，Linux 就是依靠互联网才迅速发展了起来，Linux 具有强大的网络功能也是自然而然的事情。它可以轻松地与 TCP/IP、LAN Manager、Windows for Workgroups、Novell Netware 或 Windows NT 网络集成在一起，还可以通过以太网卡或调制解调器连接到 Internet 上。

Linux 不仅能够作为网络工作站使用，更可以胜任各类服务器，如 X 应用服务器、文件服务器、打印服务器、邮件服务器、新闻服务器等等。可以说，Linux 操作系统的网络

功能胜过了其他任何一种操作系统，连 Windows NT 也不是它的对手。

8．是完整的 Unix 开发平台

Linux 支持一系列的 Unix 开发工具，几乎所有的主流程序设计语言都已移植到 Linux 上并可免费得到，如 C、C++、Fortran 77、ADA、PASCAL、Modual 2 和 3、Tcl/TkScheme、SmallTalk/X 等。

9．开放的源代码

Linux 的源代码完全开放，而且任何人可以修改这些源代码，修改后的源代码可以自己使用，也可以发布到网上让所有计算机用户使用。

本丛书内容简介

丛书分为三本，分别是《Linux 的安装与使用》、《Linux 的管理与配置》和《Linux 的内核与编程》。丛书全面阐述了 Linux 的方方面面，通俗易懂，是笔者多年使用 Unix/Linux 经验的结晶。

《Linux 的安装与使用》主要面向 Linux 操作系统的初学者，从安装和使用两方面详细讲解了 Linux 的基础知识。

《Linux 的管理与配置》主要面向 Linux 操作系统的中、高级读者，详细说明了 Linux 所完成的各个功能的配置方法，尤其适合系统管理员或者网络管理员参考。

《Linux 的内核与编程》主要面向 Linux 操作系统的中、高级读者，从操作系统原理开始，以各个方面的编程方法结束，对计算机专业的学生和程序员都由很大的帮助。

本书内容简介

本书面向 Linux 操作系统的中、高级读者，假设读者对操作系统原理或软件开发有一定了解，但不熟悉 Unix/Linux 环境。

本书分为两大部分。

第一部分： Linux 内核原理

本部分介绍 Linux 是如何实现的。

第 1 章介绍软件的一些基本知识，包括什么是计算机语言、什么样的程序才叫操作系统，以及操作系统用到的一些数据结构的实现方法。

第 2 章详细描述什么是内存管理、虚拟内存的概念以及 Linux 怎样进行页的分配、释放和交换。

第 3 章介绍 Linux 的进程概念、进程如何调度、进程与文件、虚拟内存的关系，还有可执行文件的格式。

第 4 章讲到了 Linux 进程间通信的方法，包括信号、管道、Sockets、信号量、消息队列和共享内存等。

第 5 章介绍 Linux 的 PCI 管理部分，包括 PCI 地址空间：I/O 和内存地址、PCI 配置头、PCI-ISA 桥和 PCI-PCI 桥、以及 PCI 总线的初始化工作。

第 6 章介绍 Linux 的中断概念和中断的实现，包括可编程中断控制器、中断处理数据结构的初始化和中断的处理方法。

第 7 章介绍 Linux 设备驱动器的实现，包括 DMA、设备驱动器怎样使用内存、字符设备和块设备的区别、IDE 和 SCSI 硬盘以及网络设备的初始化。

第 8 章介绍 Linux 怎样维护它支持的文件系统中的文件，同时描述虚拟文件系统并解释 Linux 内核怎样支持实际的文件系统。

第 9 章介绍 Linux 怎样支持 TCP/IP 网络协议，包括 INET Socket 层、IP 层和 IP 路由。

第 10 章描述一些内核需要支持的通用任务和机制，这样内核的其他部分能高效地工作，包括队列、计时器和锁等。

第 11 章介绍 Linux 内核怎样只在它需要时动态加载功能，例如文件系统，包括加载和卸载两部分。

第 12 章描述你应该在哪里开始查找特定内核功能的源代码，这是内核源代码的一个浏览，它们怎样编排和你可能在哪里找到特定的源代码。

第 13 章列出 Linux 使用的和本书描述的主要数据结构。

第二部分：编程方法

这里介绍 Linux 附带的一些小程序以及 Linux 的窗口系统：X Window。

第 14 章介绍 Linux 中的一个编程工具：GAWK，它以很短的程序，轻易地完成对文本文件里的数据进行修改、对比和抽取等处理。

第 15 章介绍 Linux 中的另一个编程工具，也是最重要的工具：GNU C，包括什么是 C 语言、它的重要性、gcc 和 gdb 的使用方法以及 Linux 的编程基础。

第 16 章介绍一些系统调用，包括文件和记录的加锁、任务控制、符号连接、用户和组几个部分。

第 17 章介绍多进程编程的基本概念、多进程编程的特点、进程的启动和结束、子进程、与进程相关的一些系统调用和信号的编程处理。

第 18 章介绍网络编程要注意的一些方面，包括什么是 Socket、Socket 的类型、Socket 的地址、文件名字空间和 Internet 名字空间、Socket 的各个层面、Inetd 精灵进程以及一些系统调用。

第 19 章介绍一种广泛使用的 X-Window 编程工具 GTK。GTK 建立在 GDK(GIMP Drawing Kit)的上层，基本上将 Xlib 功能包装了起来。

目　录

第一部分　Linux 内核原理

第1章　软 件 基 础

程序是执行特定任务的计算机指令集。程序可以用非常低级的汇编语言编写，也可以使用高级的、与机器无关的高级语言（例如 C）编写。操作系统是一个特殊程序，它允许用户运行类似电子表格和字处理器的应用程序。本章是一些基本的程序原理和操作系统的目的和功能的简介。

1.1　计算机语言

1.1.1　汇编语言

CPU 从内存得到指令然后执行。机器指令精确地告诉计算机该做什么。十六进制数 0x89E5 是一条 Intel 80486 指令，用于拷贝 ESP 寄存器的内容到 EBP 寄存器。汇编语言编译器是一种早期的软件工具，用来把人类能读懂的源文件汇编成机器代码。汇编语言依赖于特定的微处理器。Linux 内核的很少一部分由汇编语言编写，这些部分是高效的并且与特定的微处理器相关。

1.1.2　C 程序语言和编译器

用汇编语言写大型程序是相当困难的，也是很费时的，并且这些程序是不可移植的。而使用 C 这样的与机器无关的语言就好多了。编译器编译 C 程序为汇编语言，再生成特定的机器代码。好的编译器生成的汇编指令与使用汇编语言编写的程序基本一样高效。Linux 内核的大部分是用 C 语言编写的。C 代码被组织进程序，每一个执行一个任务。程序能返回由 C 支持的任何值或数据。大型程序像 Linux 内核包含许多的 C 模件，每个模件由程序和数据结构组成。这些 C 源代码模件组合在一起完成诸如文件系统这样的功能。

C 支持许多类型的变量，由标识符名代表的变量位于内存中。一个叫 x 的变量，它的内存地址是 0x80010000；还有一个指针：px，它指向 x，地址可能是 0x80010030。那么，px 的值是 0x80010000，这是变量 x 的地址。

C 允许绑定相关的变量成为数据结构。例如，

```
struct{
  int i;
```

```
    char b;
} my_struct;
```

是一个叫 my_struct 的数据结构，它包含两个元素，一个叫 i 的整型变量（32 位数据），另一个叫 b 的字符型变量（8 位数据）。

1.1.3 连接器

连接器把一些目标模件和库连接成一个单一的、连贯的程序。目标模件是汇编器或编译器生成的机器代码，包含了可执行的机器代码和数据，还有供连接器使用、将这些目标模件组合成程序的信息。例如一个模件可能包含程序的数据库功能，另一个模件包含该程序的命令行参数功能。Linux 内核是一个大程序，并且由许多目标模件组成。

1.2 什么是一个操作系统？

如果说计算机的硬件是它的心脏，那么软件是它的灵魂。操作系统是系统程序的集合，它允许用户运行应用程序。操作系统抽象了一个实际的硬件系统，使用户运行在一个虚拟的机器上。大多数 PC 能运行一个或多个操作系统，这些操作系统看起来和感觉上都相差很远。Linux 由很多分离的功能块组成，其中一部分是内核，但是如果没有库和外壳（shell）也是无用的。

为了理解操作系统是什么，看一下输入下面这个简单命令会发生什么事情：

```
$ ls
Mail    c       images      perl
docs    tcl
$
```

"$"是登录外壳（login shell，此例中指 bash）输出的提示符。意思是等待你（用户）输入一些命令。输入"ls"，键盘驱动器意识到有字符被输入。键盘驱动器把它们传给 shell，shell 查找这个可执行命令。找到了，是/bin/ls，内核服务把 ls 可执行命令载入虚拟内存并且开始执行它，ls 调用内核的文件子系统发现有哪些有效的文件。文件系统可能使用被缓存的文件系统信息或使用磁盘驱动器从磁盘读取该信息，甚至可能导致网络驱动器从远方机器读取远方文件的信息（文件系统能通过 Networked File System——NFS 被远程安装）。不论信息位于哪里，ls 输出该信息，视频驱动器再把它显示在屏幕上。

以上这些相当复杂，但是这说明甚至最简单的命令也需要操作系统的不同部件协同作业。

1.2.1 内存管理

操作系统不得不把安装的有限内存做欺骗性处理，能被程序认为有很多的内存。这部分视觉上的内存是虚拟内存。系统把内存分成许多页，在系统运行时把这些内存页与硬盘进行交换。软件不知道这样的处理，因为还有其他欺骗性措施：多进程。

1.2.2 进程

进程能被理解为活动的程序，每个进程是分离的实体，正在运行一个特定的程序。如果在 Linux 上查看进程，会发现有相当多的进程。例如可以使用 ps 命令显示当前进程。如果

系统有许多 CPU，那么每个进程能（至少在理论上）运行在一个不同的 CPU 上。不幸的是，只有一个 CPU，操作系统便又进行另一个欺骗，让每一个进程只能连续运行很短的周期。这个时间周期就是时间片。这个欺骗就是多进程或调度（multi-processing or scheduling），欺骗每一个进程，使它们以为只有一个进程。操作系统给每个进程一个分离的地址空间。

1.2.3 设备驱动器

设备驱动器是 Linux 内核的主要部分，像操作系统的其他部分，它们运行在一个高优先级的环境中，如果它们出现错误会导致灾难。设备驱动器控制操作系统和它们控制的硬件设备之间的交互作用。例如，当向 IDE 磁盘写块数据时，文件系统使用通用的块设备接口。驱动器关心设备细节并确保特定的事情发生。设备驱动器特定于设备的控制芯片。例如，如果系统有 NCR810 SCSI 控制器，你就需要有 NCR810 SCSI 驱动器。

1.2.4 文件系统

Linux 与 Unix 相同，它不把系统中分离的文件系统按设备标识访问（例如驱动器号或驱动器名），而是混合在单个树形结构中。Linux 把安装设备的目录加进文件系统树中，例如 /mnt/cdrom。Linux 的重要特性之一是它支持许多不同的文件系统。它的文件系统管理很灵活并能和其他很多操作系统共存。Linux 最常用的文件系统是 EXT2。

文件系统方便用户浏览系统中保持在磁盘上的文件和目录，而不用关心它们所在的物理设备的特性。Linux 透明地支持很多不同的文件系统（例如 MS-DOS 和 EXT2）并把所有的文件和文件系统集合进一个虚拟的文件系统中。因此，一般来说，用户和进程不需要知道文件系统的种类，只要使用它们。

块设备驱动器隐藏了物理块设备类型之间的不同（例如 IDE 和 SCSI）。设备的块尺寸也可以不同，例如软盘设备是 512Byte 而 IDE 硬盘是 1024Byte。不论所在的设备有何差别，一个 EXT2 文件系统是相同的。

1.3 内核数据结构

操作系统必需保持大量系统当前状态的信息。而且系统状态改变时，这些数据结构也要同时改变。例如，当一个用户登录进系统时可能要建立一个新进程。内核必须增加一个数据结构来代表这个新进程同时把它与代表系统中所有其他进程的所有数据结构相联接。

通常这些数据结构位于物理内存中，并由内核和内核子系统访问。数据结构包含数据和指针，其他数据结构的地址或程序的地址。总的来说，由内核所使用的数据结构看起来很混乱。每一个数据结构都有一个用途，也许它们中的一些由内核子系统使用，其实它们比看起来更简单。

应该知道 Linux 内核怎样管理这些不同功能的数据结构并如何使用它们。本书基于 Linux 内核描述数据结构，讲述每个内核子系统的算法术语，实现方法和内核数据结构的用法。

1.3.1 链表

Linux 使用一些软件工程技术把它的数据结构联接在一起。在许多场合，它使用联接的（Linked 或 Chained）数据结构。每个数据结构描述一个实例（Instance）或正在发生的一

些事情，例如一个进程或一个网络设备，内核必须能够找到所有的这些实例。在一个链表里，一个 root 指针包含第一个数据结构的地址或元素（Element），在该列表里每一个数据结构包含列表中下一个元素的地址，最后一个元素的下一个指针（Nest Pointer）应该是 0 或 NULL 来表示它是该列表的结尾。在一个双向联接（Doubly Linked）表中，每一个元素包含了指向列表中下一个元素和指向前一个元素的指针。使用双向列表能更容易地在列表的中间位置增加或删除元素。这是一个典型的操作系统交换过程：对于 CPU 循环的内存访问。

1.3.2 哈希表（Hash Tables）

使用链表联接数据结构非常容易，但是遍历（navigating)链表效率太低。如果你查找部分元素，可能需要查遍整个列表。Linux 使用另一个技术，hashing 来提高效率。一个哈希表是一个指针数组或指针矢量（array or vector of pointers），数组或矢量的元素在内存中简单地连续排列。数组通过索引（index）进行访问，索引是数组的偏置。

哈希表是一个数据结构的指针数组，它的索引是一些源于那些数据结构的信息。如果你有描述一个乡村人口的数据结构，你可以使用人的年龄作为索引，这样建立的哈希表可以用来查找一些人的数据。一个乡村中的好多人有相同的年龄，所以哈希表的一个指针是一些数据结构的联接，这些数据结构有相同的年龄。搜索这些较短的链比搜索所有的数据结构要快得多。

通常哈希表能加快数据结构的访问，Linux 经常使用哈希表实现缓存（Caches）功能。缓存用来保存需要快速访问的信息，并且通常是有效信息的子集。数据结构被存入缓存并保持在那里是因为内核经常访问它们。使用缓存有一个缺点，它们比链表和哈希表使用起来更加复杂。如果数据结构能够在缓存中被找到（也叫做 Cache Hit），一切都好说。如果找不到，要搜索所有相关的数据结构，如果该数据结构确实存在，它必须被加进缓存中。新的数据结构加进缓存中，旧的数据结构就要被移出，Linux 决定哪些将要被移出，危险的是移出的数据结构也许是 Linux 下一个要用到的。

1.3.3 抽象接口（Abstract Inferfaces）

Linux 经常抽象它的接口。一个接口是程序和数据结构的集合，用特定的方法操作。例如所有的网络设备驱动器不得不提供一定的操作特定数据结构的程序。可以使用通用层的代码代替较低层的代码提供的服务或接口（Services or Interfaces）。网络层是通用的，而支持它的特定设备代码使用标准接口。

较低层经常在上一层启动（Boot）时登记（Register）它们自己。登记一般指数据结构加进链表。例如每个文件系统在内核启动时把它自己登记在内核中，如果在文件系统还没被使用前先使用它，你可以看见已登记的文件系统看上去的形式是/proc/filesystems 文件。登记的数据结构经常包含函数的指针，这些指针是软件函数的地址，函数用来执行一定的任务。还是使用文件系统的例子，每个文件系统传给 Linux 内核，作为它的登记项的数据结构中，一定包含这样一个文件系统程序地址，该程序被调用来查看文件系统是否已安装。

第2章 内存管理

内存管理子系统是操作系统最重要的一个部分。在早期的计算机中，也需要大量的物理内存。为超越物理内存的限制，开发了虚拟内存。虚拟内存看起来比物理内存大，并由正在完成的进程所共享。

虚拟内存不仅把你的计算机内存变大。内存管理子系统还提供：

- 大的地址空间　操作系统使系统看起来安装了比实际安装的内存更大的容量。虚拟内存能比系统中的内存大许多倍。
- 保护　系统中的进程占用它们自己的虚拟内存地址空间。这些虚拟内存地址空间彼此完全分离，所以进程运行的程序不会影响另一个。而且，硬件虚拟内存机制允许把内存区域保护起来，防止被重写。这可以保护代码和数据不被恶意程序重写。
- 内存镜像　内存镜像把图形和数据文件映射到进程地址空间。通过内存镜像，文件内容被直接链接到进程的虚拟地址空间。
- 物理内存分配　允许系统中每一个运行的进程很好地得到系统物理内存。
- 共享虚拟内存　虚拟内存允许进程有分离（虚拟）的地址空间，有几种方法使用共享内存。例如系统允许几个进程同时调用 bash（一种 Linux shell）。强于在每个进程的虚拟地址空间中使用 bash 的拷贝，更好的方法是在物理内存中有一个拷贝每一个调用 bash 的进程共享它。动态库是另一个在数个进程间共享可执行代码的例子。共享内存也能用于进程间通信机制（Inter Process Communication--IPC），两个或者更多的进程通过它们的公用内存交换信息。Linux 支持 Unix System V 的共享内存 IPC 机制。

2.1 虚拟内存的抽象模型

在认识 Linux 支持虚拟内存的方法之前，认识其不含太多细节的抽象模型是很有好处的。

在进程执行一个程序时，它从内存读取指令并进行译码。在翻译指令过程中，它可能需要得到或者存储内容在内存中的位置。处理器然后执行指令且移到程序的下一条指令。在这个过程中处理器总是通过访问内存来得到指令或者得到和访问数据的。

在一个虚拟内存系统中，所有这些地址都是虚拟地址而不是物理地址。处理器通过操作系统维护的表格中的信息把这些虚拟地址转换成物理地址。

为了使这个转换变得容易，虚拟内存和物理内存被分成便于计算的尺寸块叫做页（Pages）。这些页有相同的尺寸，它们不是必须的，但是如果不这样做，系统将很难管理。Alpha AXP 系统上的 Linux 使用 8KB 的页而 Intel x86 系统上的 Linux 使用 4KB 的页。每一个页使用唯一的标号，叫页架构号（Page Frame Number--PFN）。在这种页模型中，一个虚拟地址由两个部分组成：一个偏移量和一个 VPFN。如果页尺寸是 4KB，虚拟地址的低 12 位包含偏移量，12 位及其以上的位是 PFN。处理器每次遇到一个虚拟地址它必须从中抽取偏移量和 PFN。处理器把虚拟 PFN 转换成物理的然后访问物理页中正确的偏移位置。处理器使用页表（Page Tables）完成该功能。

图 2-1 显示了两个进程的虚拟地址空间。进程 X 和进程 Y，每一个都有自己的页表。这些页表把每个进程的虚拟页镜像成内存中的物理页。图 2-1 中显示，进程 X 的虚拟 PFN 0 被镜像成物理内存 PFN 1，进程 Y 的虚拟 PFN 1 被镜像成物理内存 PFN 4。每一个理论上的页表的入口包含以下信息：

- 有效标志，指示这个页表的入口是否有效。
- 物理 PFN 的入口是否可描述的。
- 访问控制信息，描述该页怎样使用，它能否被写，它是否包含可执行代码。
- 被访问的页表使用 VPFN 作为一个偏移量，虚拟的 PFN 5 是表的第六个元素（0 是第一个元素）。

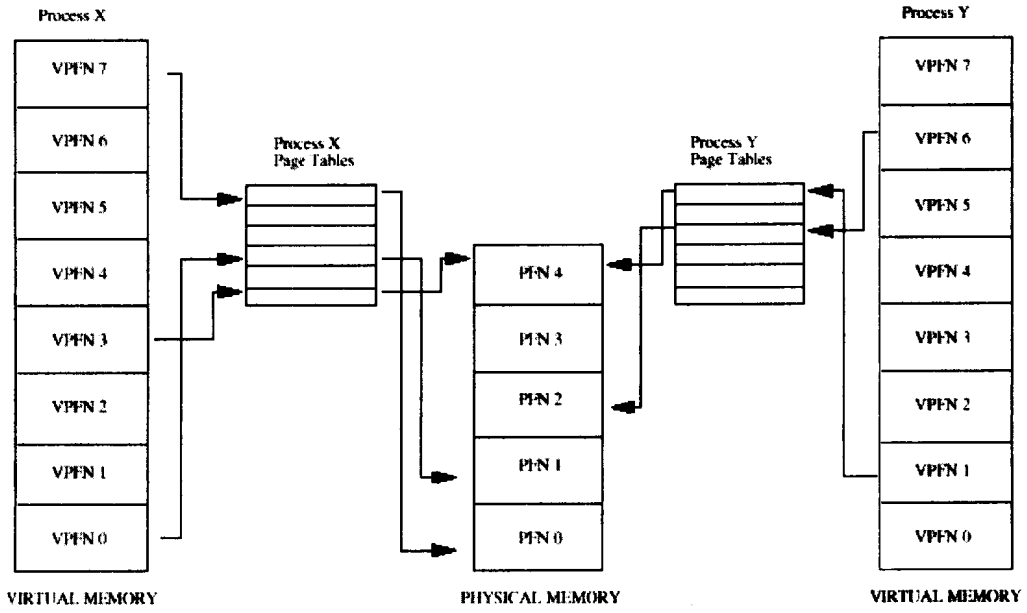

图 2-1 虚拟地址到物理地址镜像的抽象模型

为了把一个虚拟地址转换成一个物理地址，进程必须首先计算出虚拟地址 PFN 和虚拟页内的偏移量。通过使页尺寸乘以 2（使用掩码和移位算法能很容易地得到这个值），再看一下图 2-1，假设页尺寸有 0x2000（8192）Byte，地址 0x2194 在进程 Y 的虚拟地址空间内，则处理器将把这个地址转换成偏移量 0x194 和 VPFN 1。

进程使用 VPFN 作为一个索引用来检索它的页表入口。如果页表入口所在的偏移量是有效的，处理器降低这个入口的物理 PFN。如果入口是无效的，处理器将访问一个它的虚拟内存中不存在的区域，在这种情况下，处理器不能分解这个地址，将由操作系统解决这个问题。

处理器怎样通报操作系统某个进程已经无效地尝试访问一个由处理器转换的虚拟地址？不论处理器如何递交这个错误，操作系统都知道产生了一个页故障（Page Fault），并且知道页故障的故障虚拟地址和产生原因。

假设有一个有效的页表入口，处理器得到物理 PFN 并且乘以页尺寸得到物理内存页的内存基址。最后，处理器加上偏移量来获得它需要的指令或数据。

再次使用上面的例子，进程 Y 的 VPFN 1 被镜像到物理页 PFN 4，它起始于地址 0x8000，加上偏移数 0x194，最终的物理地址是 0x8194。

由于把虚拟地址镜像到物理地址，虚拟内存能以任何顺序镜像到系统物理页。

2.1.1 按要求分页（Demand Paging）

由于物理内存比虚拟内存少，操作系统必须高效使用物理内存。一个节省内存的方法是只加载可执行程序正在使用的虚拟页。例如，一个数据库程序可能被运行来查询一个数据库。不是数据库的所有数据都需要被加载入内存，只需要被查询到的数据。如果该数据库查询是一个“搜索”（Search）查询，它不会加载数据库程序中处理增加新记录的代码。这种只把当前处理的虚拟页加载进内存的技术叫按要求分页。

当一个进程尝试访问一个当前不在内存中的虚拟地址时，处理器不能发现所涉及的虚拟页中的虚拟地址。例如，在图 2-1 中，进程 X 的 VPFN 2 没有页表入口，如果进程 X 尝试读 VPFN 2 中的一个地址，处理器不能把这个虚拟地址转换成物理的。处理器通报操作系统产生了一个页故障。

如果该故障的虚拟地址是无效的，即进程已经尝试访问了一个它不应该拥有的地址。可能程序在某些方面已经产生了错误，例如在内存中访问一个随机地址，此时操作系统将终止它，以保护系统中其他进程。

如果该故障的虚拟地址是有效的，但是它指向的页现在不在内存中，操作系统必须从磁盘映像中把正确的页加载入内存。磁盘访问需要较长时间，所以进程需要等待以得到该页。如果这时有其他进程能够运行，操作系统将选择其中一个来运行。得到的页将被写入一个空的物理页架构，这个 VPFN 的入口将被加到该进程的页表中。这个进程然后能够从发生内存故障的机器指令处开始重新执行。这一次再进行虚拟内存访问，处理器能把虚拟地址转换成物理地址，进程继续运行。

Linux 使用按要求分页来把可执行的映像加载进进程虚拟内存。只要一个命令被执行，包含命令的文件被打开，内容被镜像成进程虚拟内存。这通过修改描述这个进程内存镜像的数据结构来实现，叫做内存镜像（Memory Mapping）。无论如何，只有映像的第一部分被实际加载进物理内存，映像的剩余部分保留在磁盘上。映像执行时，它会产生页故障，Linux 使用进程内存镜像来确定映像的哪部分被加载进内存执行。

2.1.2 交换（Swapping）

如果程序需要把一个虚拟页加载入内存，但是已经没有足够的剩余内存页，操作系统必须把该页载入内存同时把另一页置换出物理内存。

如果从物理内存置换出的页来自于一个映像或数据文件，并且如果没有被写过则不必被存储。另一方面，如果进程再次需要该页，它还可以重新从映像或数据文件载入内存。

无论如何，如果页已经被修改了，操作系统必须保存该页的内容这样它才能在以后再被访问。这一类的页被叫做 Dirty Page，当它从内存中被交换出，它被存入一个经过排序的叫做交换文件（Swap File）的特殊文件中。访问交换文件相对于处理器与内存之间的访问速度是比较慢的，操作系统还要在程序需要的时候把修改过的页加载入内存。

如果被用来决定交换哪些页的算法（交换算法）是无效的，产生一个 Thrashing 状态。这时，长时间没有变化的页被写入磁盘，然后再被读入，这样操作系统会太忙以至于不能完成很多实际工作。例如，图 2-1 中的物理 PFN 1 经常被访问，则它不是一个与磁盘进行交换的好的候选页。一个进程当前正在使用的页的集合被叫做工作集（Working Set）。一个有效